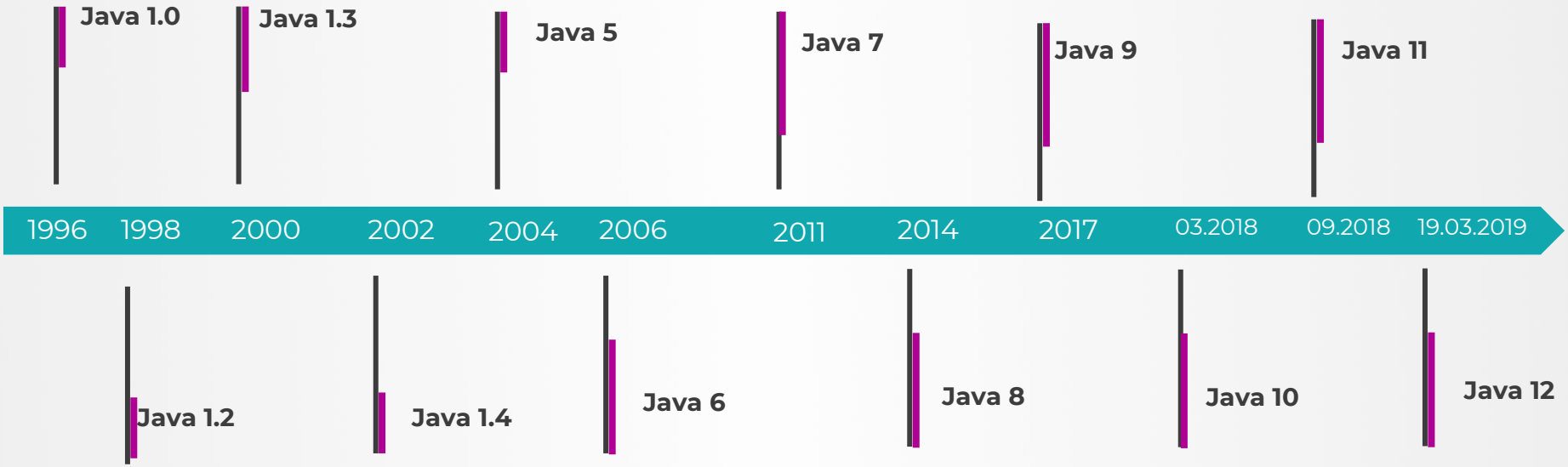
Java 8

Maciej Koziara

Java releases



Kompatybilność wsteczna

Zapewnienie, że aplikacja napisana w starszej wersji języka, poprawnie uruchomi się z wykorzystaniem nowej wersji Javy.

Zalety: poczucie stabilności, łatwość podbijania wersji Javy w projektach

Wada: powolny rozwój języka

Dlaczego Java 8?

- Nowy, funkcyjny paradygmat programowania
- Składnia lambda - skrócony zapis funkcji
- Stream API - łatwe przetwarzanie kolekcji
- **Optional** - początek walki z NPE
- **DateTime API** - porządna obsługa dat i czasu w Javie

Let’s get started!

Immutability

// Gwarancja, że obiekt raz utworzony nigdy nie zostanie zmieniony.
// Próba zmiany obiektu tworzy nowy, z odpowiednio zmienionymi wartościami.
// Zarówno **DateTime API** jak i **Optional API** są Immutable

Dates

// W systemach komputerowych datę zapisuje się jako liczbę sekund które upłynęły od 01.01.1970 r. Pomimo iż jest to wydajne, nie jest najwygodniejsze do pracy.
// **LocalDate** i **LocalDateTime** zostały wprowadzone do Javy, aby uprościć obsługę dat
// Zawsze używaj **LocalDate** lub **LocalDateTime** gdy pracujesz z datami. Nigdy starych metod używających klas **Date** lub **Calendar**.

Dates - podstawowe operacje

// *now()* - tworzy datę reprezentującą aktualny dzień
// *addDays(), addWeeks() itp.* - manipulują datą
// *parse()* - tworzy datę ze Stringa zapisanego w podanym formacie
// *format()* - konwertuje datę na String o podanym formacie
// **ChronoUnit.between()** - pozwala na liczenie okresu między datami

Dates - formatowanie

// **DateTimeFormatter** - używany do formatowania **LocalDate** oraz **LocalDateTime**

Operatory:
y - rok
M - miesiąc
d - dzień

Dates - formatowanie

// **yyyy-MM-dd -** 2018-07-11
// **yy-dd-MMMM -** 2018-11-July
// **yyyy-MM-dddd -** 2018-07-Wednesday
// **yyyy-MMM-ddd -** 2018-Jul-Wed

// Możesz użyć klasy **Locale** aby wskazać język, który powinien zostać wykorzystany do wyświetlania nazw miesięcy i dni tygodnia.

Problemy z nullem

// Brak pewności czy dany obiekt może być nullem czy nie -> nigdy nie wiadomo, czy można go bezpiecznie użyć
// Zazwyczaj nulle pojawiają się tylko w szczególnych przypadkach, przez co namierzenie problemu może być trudne i zająć dużo czasu
// Java 8 wprowadziła **Optional**, po to aby ułatwić programistom pracę z nullami

Optional - podstawy

// Reprezentuje istniejącą wartość lub jej brak (null)
// Jest generyczny - przechowuje wartości różnego typu
// Posiada dwa stany:

- **present** -> wartość istnieje i można jej bezpiecznie użyć
- **empty** -> wartość nie istnieje

Optional API

// *of()* - tworzy **Optional** z podaną wartością
// *empty()* - tworzy pusty **Optional**
// *ofNullable()* - zależnie czy podana wartość jest nullem, tworzony jest **empty** lub **present Optional**
// *isPresent(), isEmpty()* - sprawdza w jakim stanie jest optional
// *get()* - pobierz wartość z Optionala
// *orElse()* - pobierz wartość z Optionala lub zwróć domyślną jeśli wartość nie istnieje